LA-UR-21-?????

Approved for public release; distribution is unlimited

Matrix-Based Process Modeling in Microsoft Excel

John Sherwood

January 19, 2021

Los Alamos NATIONAL LABORATORY EST. 1943

NNSA National Nuclear Security Administration

Table of Contents

1. Methodology 1
  1.1. Towards the Process Matrix 2
  1.2. Final Output 3
  1.3. Scaling Vector 3
  1.4. Auxiliary Matrices 3
  1.5. Putting it Together 5
  1.6. Towards a Stochastic Model 5
  1.7. Assumptions & Limitations 5
2. Implementation in Microsoft Excel 6
  2.1. Assembling the Process Matrix 6
  2.2. Taking the Inverse 8
  2.3. The Scaling Vector 8
  2.4. The Auxiliary Matrix 9
  2.5. Implementing Basic Stochasticity 9
Conclusion 10
Works Cited 11

This page intentionally has no information

Introduction

The purpose of this document is to describe the theory and methodology of matrix-based process modeling and demonstrate its use through a generic process model within Microsoft Excel. This type of model allows a modeler to quickly query various metrics, such as total production time, for a specified production run. These matrix models tend to be quick to build and use, but come with certain limitations and are more suited for smaller processes.

Matrix-based methods have been used in a variety of fields for modeling linkages between processes. Within the field of economics, Input-Output (IO) models of the economy were developed by Wassily Leontief during WWII to better track supply chains. These first IO models were used in part for war planning [1]. Outside of economics, the mathematical discipline of graph theory uses adjacency matrices which describe how different nodes (processes, people, or items) are linked by edges (product flows, information flows, or relationships) [2]. These network models can be used to trace how an infectious disease spreads within a community, given relationships or links between people. Finally, the field of Life Cycle Assessment (LCA), which models the environmental impacts of products and processes, has been using a matrix-based process modeling approach for the last two decades [3].

While matrices have been used in Operations Research (e.g., for the traveling salesman problem [4]) and in chemical process modeling [5], I am unaware of their use specifically to model production processes.

Matrix-based process modeling provides a concise and relatively simple platform with which to track inputs, outputs, and various flows through a production process. It is not a simulation, but rather an analytical technique. As such, there are inherent limitations that the modeler ought to be aware of – these are further described in Section 1.7.

This report consists of two main sections. Section 1 describes the methodology, while Section 2 demonstrates the implementation of a generic process model in Excel.

1. Methodology

A generic process flow shown in Figure 1 will be analyzed and used throughout this report. This process contains a few elements that render modeling-by-hand challenging, including the existence of waste streams (on *Process 2* and *6*), a rework or recycling stream (from *Process 4* to *6*), and pieces of equipment used across multiple process steps. These interdependencies prohibit a modeler from directly counting the number of cycles each process encounters for a specified output. The primary goal of a matrix-based methodology is to capture these interdependencies and the iterative nature of rework in a concise form. This matrix can be used to find the exact number of cycles for each process to produce some number of products. Then, the number of process cycles can be used to study auxiliary metrics of the process, such as equipment usage and worker-hours.

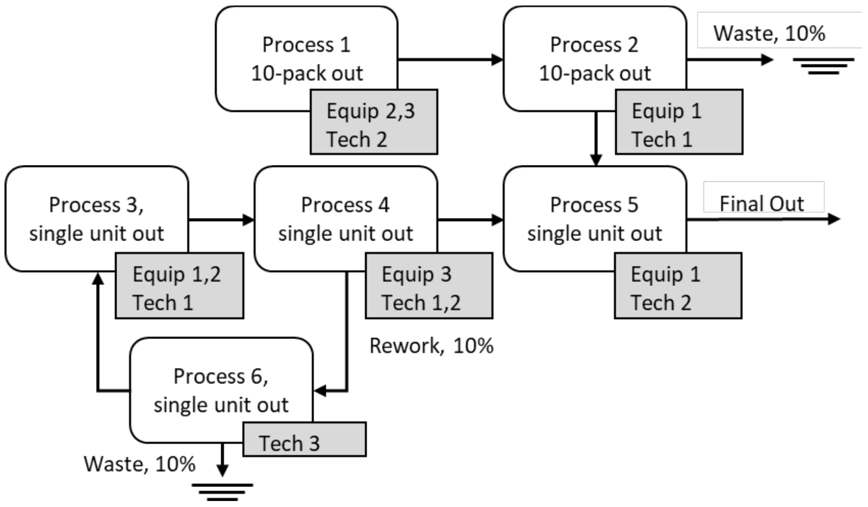


Figure 1. Generic process flow. Note wastes, rework, and multiple pieces of equipment and technicians.

1.1. Towards the Process Matrix

Before looking at the process as a whole, the unit processes need to be quantified. Ultimately, I will link these unit processes together in a matrix, but the initial approach is to build vectors for each individual process. Consider *Process 1*, which has no inputs and outputs a product. The product of *Process 1* will be Pd_1. The product of *Process 2* will be Pd_2, and so on. Each process produces a single product that may be used in one or more processes, or may be delivered to the final customer.

I will represent these product flows through a vector for each process. The vectors for *Process 1* and *Process* 2 are:

$$P_1 = \begin{pmatrix} Pd_1 \\ Pd_2 \\ Pd_3 \\ Pd_4 \\ Pd_5 \\ Pd_6 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \qquad P_2 = \begin{pmatrix} -1 \\ 0.9 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{1.1}$$

Here, *Process 1* has no inputs from the other processes, and outputs one unit of its product. *Process 2* takes in one unit from *Process 1*. Because of the 10% waste stream, it can only output 0.9 units of Pd_2 for every unit of Pd_1 input.

The remaining vectors for the other unit processes are:

$$P_3 = \begin{pmatrix} Pd_1 \\ Pd_2 \\ Pd_3 \\ Pd_4 \\ Pd_5 \\ Pd_6 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ -0.1 \end{pmatrix}, \qquad P_4 = \begin{pmatrix} 0 \\ 0 \\ -1 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \qquad P_5 = \begin{pmatrix} 0 \\ -0.1 \\ 0 \\ -1 \\ 1 \\ 0 \end{pmatrix}, \qquad P_6 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ -1 \\ 0 \\ 0.9 \end{pmatrix} \tag{1.2}$$

There are a few things to note here. I start with describing the rework stream. *Process 4* outputs 1 unit of product every cycle, despite two different output streams. This differs from the waste stream in *Process 2*, because the rework stays within the technical system. Therefore, *Process 4* outputs 1 rather than 0.9. *Process 6*, a rework or inspection step, outputs 0.9 units for each input from *Process 4* that it receives.

Process 3 takes in -0.1 of Pd_6 for each unit of output, because the rework stream was specified as 10% of the throughput. Finally, *Process 5* handles a unit conversion – only 1/10th of Pd_2 (a ten-pack) is needed to produce one unit of the final output.

These vectors can be stacked together to form a matrix, often designated the $\mathbf{A}$ matrix:

$$\mathbf{A} = [P_1|P_2|P_3|P_4|P_5|P_6] = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0.9 & 0 & 0 & -0.1 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & -1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & -0.1 & 0 & 0 & 0.9 \end{bmatrix} \quad (1.3)$$

The $\mathbf{A}$ matrix is usually referred to as the "technical" matrix or "process" matrix [3] [6]. It represents, and only represents, the outputs and linkages of each process running for one cycle. It does not directly account for waste, energy or material inputs, production times, etc. And, it does not capture the number of cycles needed to produce a specified final output.

1.2. Final Output

Another vector specifies the desired final output. This vector, F, specifies the number of units of each product delivered to the customer. For example, 100 finished products (Pd_5) would be represented as:

$$F = \begin{pmatrix} Pd_1 \\ Pd_2 \\ Pd_3 \\ Pd_4 \\ Pd_5 \\ Pd_6 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 100 \\ 0 \end{pmatrix} \quad (1.4)$$

This vector, F, is often called the final demand vector. Note that, if I wanted to remove 20 units of Pd_2 for destructive testing during the production process, I could change the second entry of F to 20 to account for the additional demand.

1.3. Scaling Vector

The final piece of this puzzle is to determine the number of cycles needed to produce F. If $\mathbf{A}$ represents the outputs and requirements of processes for one cycle, then I could multiply $\mathbf{A}$ by a number of cycles for each process to determine the final output, F. That is,

$$\mathbf{A}S = F \quad (1.5)$$

Where S is a scaling vector representing the number of cycles for each process. However, as noted in Equation 1.4, F is specified and S is to be solved for. Therefore,

$$\mathbf{A}^{-1}F = S \quad (1.6)$$

Equation 1.6 is the core component of this matrix-based methodology. To reiterate, S represents the number of cycles that each of the six processes must run to produce the final output. In this example,

$$S = (11.\bar{1} \quad 11.\bar{1} \quad 112.5 \quad 112.5 \quad 100 \quad 12.5)^T \quad (1.7)$$

1.4. Auxiliary Matrices

It is powerful and useful to know the number of cycles for each process to meet a final demand, particularly with auxiliary data in the correct format. If a modeler collects data on process runtime per unit output, this can be used with the scaling vector to determine the total runtime needed to produce the final demand. For example:

$$\begin{gathered} P_{runtime} = (60 \quad 120 \quad 240 \quad 240 \quad 15 \quad 20)\ [minutes] \\ P_{runtime} \cdot S = 57{,}750\ [minutes] \end{gathered} \tag{1.8}$$

Here, $P_{runtime}$ is a vector representing each process's runtime. The column number represents the process number, as in the $\mathbf{A}$ matrix. The total runtime to produce 100 final units is 57,750 minutes.

Other vectors could be added to the analysis, representing equipment operating times, labor hours, mass flow, radiation exposure, etc. Any metric that occurs each time a process is cycled can be tabulated here. These vectors can be combined into what is known as the $\mathbf{B}$ matrix, seen in Equation 1.9. The $\mathbf{B}$ matrix is typically called the environmental matrix, intervention matrix, or auxiliary matrix. This tracks environmental factors associated with a process: anything that is not the flow of production units.

$$\mathbf{B} = \begin{bmatrix} P_{runtime} \\ Equip1_{runtime} \\ Equip2_{runtime} \\ Equip3_{runtime} \\ Tech1_{labor} \\ Tech2_{labor} \\ Tech3_{labor} \end{bmatrix} = \begin{bmatrix} 60 & 120 & 240 & 240 & 15 & 20 \\ 0 & 120 & 120 & 0 & 15 & 0 \\ 30 & 0 & 120 & 0 & 0 & 0 \\ 30 & 0 & 0 & 240 & 0 & 0 \\ 0 & 120 & 240 & 120 & 0 & 0 \\ 60 & 0 & 0 & 120 & 15 & 0 \\ 0 & 0 & 0 & 0 & 0 & 20 \end{bmatrix}, \qquad \mathbf{B}S = \begin{bmatrix} 57{,}750 \\ 16{,}333 \\ 13{,}833 \\ 27{,}333 \\ 41{,}833 \\ 15{,}667 \\ 250 \end{bmatrix} \tag{1.9}$$

The final values at the right side of Equation 1.9 represent the total process runtimes, total equipment runtimes, and total labor time in minutes. So, the labor requirements for technician three is 250 minutes. Just as Equation 1.8 multiplies runtimes by the scaling vector, multiplying the $\mathbf{B}$ matrix by the scaling vector provides the total amounts for each $\mathbf{B}$ matrix metric. Note that, while the above example uses minutes for all rows, each row may have separate units. It is possible to track, for example, runtime, water usage, electricity usage, radiation exposure, and wastes all in the same $\mathbf{B}$ matrix.

For completeness within this example, I will show the wastes associated with this process:

$$\mathbf{B} = \begin{bmatrix} Pd_2\ Waste \\ Pd_6\ Waste \end{bmatrix} = \begin{bmatrix} 0 & 0.1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.1 \end{bmatrix}, \qquad \mathbf{B}S = \begin{bmatrix} 1.1\bar{1} \\ 1.25 \end{bmatrix} \tag{1.10}$$

These wastes are separated into two rows in order to track wastes from each individual process. The total Pd_2 waste is 1.11 units, and the total Pd_6 waste is 1.25 units. These are not directly comparable because of the different product types (remember that Pd_2 represents a 10-pack of widgets). If I only cared about the total waste, rather than each individual waste stream, I could combine the waste streams into a single row of the $\mathbf{B}$ matrix as long as I convert to equivalent units. So, the Pd_2 waste would need to be multiplied by a factor of 10 to track individual units.

This technique demonstrated above of each waste stream represented as a single row could also be used to track, for example, individual process runtimes. To do so, the $P_{runtime}$ vector in Equation 1.8 would need to be diagonalized into a matrix where each row contains only the runtime for a single process. In doing so, the $\mathbf{B}$ matrix output would represent the total runtime for each process similar to how Equation 1.10 shows the total waste for each relevant waste stream.

1.5. Putting it Together

In order to generate the $\mathbf{A}$ and $\mathbf{B}$ matrices, a modeler only requires a process flow sheet and whichever metrics they wish to track per unit process. Once the matrices are constructed in a spreadsheet or programming environment (e.g. MATLAB), the modeler could change F, the final demand, at will to understand their aggregate metrics across different output scenarios. The normal usage of this model requires a specific number of final products as the input. However, Excel's goal-seek or solver tool could be used to find the number of final products that could be produced given a certain constraint, such as total runtime. As such, this matrix-based model provides a concise, quick tool with which to investigate various metrics within a production environment.

1.6. Towards a Stochastic Model

Note that this methodology is deterministic in its current form. All process variables are precisely specified as point values. Unfortunately, most process metrics exist within a range of values due to uncertainty or variations within the production environment. This variation or uncertainty can be accommodated by two different "levels" within a matrix-based model using a Monte Carlo method.

The first "level" of stochasticity that this model can handle is within the $\mathbf{B}$ matrix. Assume that process times (Equation 1.8) vary according to some distribution. I may wish to determine the average (i.e., mean) total production time. To accomplish this, the $\mathbf{B}$ matrix could be expanded by multiple rows, where each row is a different "run," or instantiation of point numbers generated by the known distribution of the runtime. That is, the $\mathbf{B}$ matrix could look like:

$$\mathbf{B} = \begin{bmatrix} P_{time}1 \\ P_{time}2 \\ P_{time}3 \\ P_{time}4 \\ P_{time}5 \end{bmatrix} = \begin{bmatrix} 60 & 120 & 240 & 240 & 15 & 20 \\ 52 & 121 & 200 & 238 & 16 & 22 \\ 73 & 119 & 220 & 232 & 14 & 32 \\ 58 & 121 & 280 & 245 & 13 & 18 \\ 65 & 122 & 245 & 241 & 15 & 25 \end{bmatrix}, \qquad \mathbf{B}S = \begin{bmatrix} 57{,}750 \\ 53{,}072 \\ 54{,}783 \\ 62{,}576 \\ 58{,}565 \end{bmatrix} \tag{1.11}$$

Then, I would calculate the mean value from the result of Equation 1.11, which is 57,349 minutes. (Note that a minimum of 100 "runs" is generally recommended to achieve stable results, depending on the range and type of distributions [3].)

It is important to note that, within this first "level" of stochasticity, the process flow represented in the $\mathbf{A}$ matrix is still deterministic. The $\mathbf{A}$ matrix contains information about the efficiency of processes, represented in reduced throughput for *Process 2* and *6*. These values are often uncertain, or at least vary, in reality.

Incorporating uncertainties into the $\mathbf{A}$ matrix is the second "level" of stochasticity. In order to incorporate this uncertainty, I would need to create several versions of the $\mathbf{A}$ matrix and invert them in order to find a different scaling vector for each version. Inverting many matrices can become computationally expensive and is best done in a programming environment such as MATLAB. Note that depending on the process flow, a small change in one process could have significant downstream effects. It may be possible to use perturbation theory to better account for uncertainty within the $\mathbf{A}$ matrix, though more research would be necessary [3].

1.7. Assumptions & Limitations

The relative simplicity of this modeling technique implies a set of assumptions and limitations, including:

1) The model represents a steady-state system that scales linearly. This means that the entire production process, and all auxiliary metrics, are the exact same for the first unit and for the millionth unit of production.

2) The model usually neglects temporal and environmental variation. The model, as currently structured, only accounts for direct process time. It does not account for lead times, production delays, stoppages, or other periodic factors that may affect a production schedule. Similarly, the model does not account for various environmental factors (e.g. equipment wear-out affecting yield rates or machining time) that may influence production times in reality. Of course, the model *could* account for these variables if the modeler factors them into the unit-process per-cycle runtimes (Equation 1.8). But note that these sources of variation would get lumped into a single average process runtime.
3) Rework and waste streams *must* occur as specified. The model cannot account for process improvement or learning during production. That is, there will be the same amount of waste from the first units of production and the millionth units of production. (It may be possible to include learning rates as a model extension, though it is outside the scope of this document.)
4) All relevant process information must be quantifiable and known; the $\mathbf{A}$ matrix must be fully specified. This means that if I build out rows and columns of the $\mathbf{A}$ matrix to represent a recycling stream, I must fill in values for those rows and columns. If I build out the $\mathbf{A}$ matrix, then later seek recycling data required to fill it in, the model will not be solvable during the interim. Data must be entered for every matrix element – either real data or placeholder values.
5) The $\mathbf{A}$ matrix must be square to be invertible in Equation 1.6. This implies that each process (each column) may only create one product (row). Chemical processes (e.g. distillation columns) occasionally produce co-products (jet fuel, gasoline, diesel), which are difficult to account for within this model structure. In this scenario, a solution may be to divide the single chemical process into multiple artificial processes, one per co-product, to maintain a square matrix. Furthermore, each product may only have one process. That is, there cannot be multiple processes that produce the same product – parallel equipment must either be modeled as a single process, or their products must be tracked individually. (The model can be extended to utilize a rectangular matrix, though it is outside the scope of this document.)

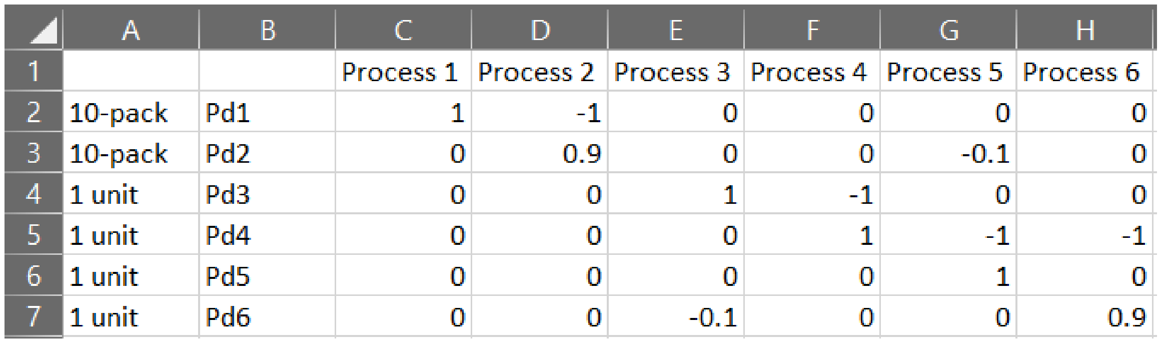
2. Implementation in Microsoft Excel

This section demonstrates how to build a matrix-based process model in Excel. Excel is an appropriate choice for implementing smaller-scale models (e.g. processes with 1-250 process steps). Excel is ubiquitous which allows for model dissemination and sharing. It also contains all the features necessary to implement a matrix model. With that said, Excel lacks version control – it can be difficult to track precise changes to a model over time. Excel can become ad-hoc and is vulnerable to human error – it is easy to inadvertently edit a cell, or to create an unorganized worksheet. Excel is also limited from advanced modeling techniques without the use of VBA code. For these reasons, MATLAB or a programming environment may be an appropriate alternative for larger matrix-based models or those requiring more features. (Though, know that MATLAB and other tools are not necessarily immune to the above issues.)

2.1. Assembling the Process Matrix

The first step within Excel is to assemble $\mathbf{A}$, the process matrix. As a reminder, each column of $\mathbf{A}$ represents a process, while each row represents a product used or made within a process. There are at least two methods of building $\mathbf{A}$ in Excel: directly and with a Pivot Table.

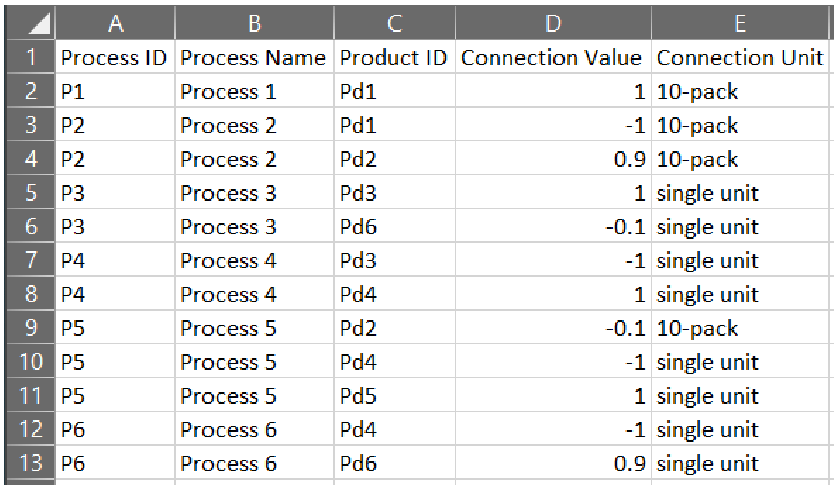
I have directly recreated the process matrix from Equation 1.3 here, in Figure 2. Note that I used *Column A* to display the units for each process in order to keep track of unit conversions (such as in *Cell G3*)



	A	B	C	D	E	F	G	H
1			Process 1	Process 2	Process 3	Process 4	Process 5	Process 6
2	10-pack	Pd1	1	-1	0	0	0	0
3	10-pack	Pd2	0	0.9	0	0	-0.1	0
4	1 unit	Pd3	0	0	1	-1	0	0
5	1 unit	Pd4	0	0	0	1	-1	-1
6	1 unit	Pd5	0	0	0	0	1	0
7	1 unit	Pd6	0	0	-0.1	0	0	0.9

Figure 2. The process matrix implemented in Excel

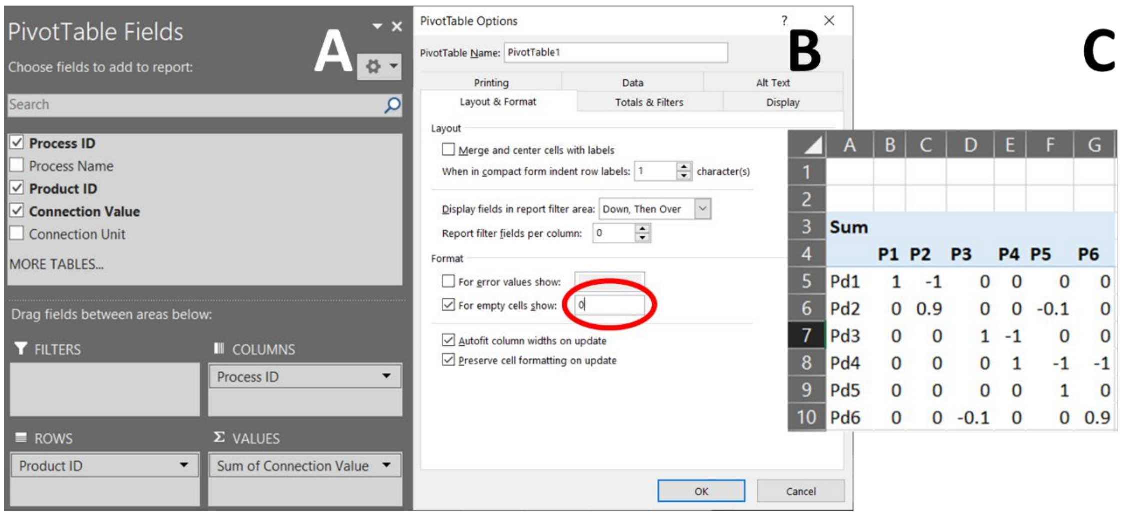
Another option to build the process matrix is through a Pivot Table and process list. Directly building a matrix may be time consuming and prone to human error for systems with many processes. Because of this, it may be beneficial to build the equivalent of an adjacency list, to use a graph theory term (the $\mathbf{A}$ matrix is nearly equivalent to a weighted adjacency *matrix*; adjacency *lists* are another form of the same information). This list is shown in Figure 3 below. Here, each row of data represents a link, or value within the process matrix. Values have associated processes (*Columns A* and *B*) and products (*Column C*). The units in *Column E* are not strictly necessary, but may help track information.



	A	B	C	D	E
1	Process ID	Process Name	Product ID	Connection Value	Connection Unit
2	P1	Process 1	Pd1	1	10-pack
3	P2	Process 2	Pd1	-1	10-pack
4	P2	Process 2	Pd2	0.9	10-pack
5	P3	Process 3	Pd3	1	single unit
6	P3	Process 3	Pd6	-0.1	single unit
7	P4	Process 4	Pd3	-1	single unit
8	P4	Process 4	Pd4	1	single unit
9	P5	Process 5	Pd2	-0.1	10-pack
10	P5	Process 5	Pd4	-1	single unit
11	P5	Process 5	Pd5	1	single unit
12	P6	Process 6	Pd4	-1	single unit
13	P6	Process 6	Pd6	0.9	single unit

Figure 3. The process list which, when used with a Pivot Table, will generate a process matrix

Selecting this list and creating a pivot table will generate the process matrix shown in Panel C of Figure 4. To use this Pivot Table in a matrix function, zeros must be added to empty cells. This option is within the Pivot Table options (available after creating the Pivot Table) and is highlighted in Panel B.



- [✓] Process ID
- [ ] Process Name
- [✓] Product ID
- [✓] Connection Value
- [ ] Connection Unit

COLUMNS: Process ID
ROWS: Product ID
VALUES: Sum of Connection Value

PivotTable Name: PivotTable1

Layout
- [ ] Merge and center cells with labels
- When in compact form indent row labels: 1 character(s)
- Display fields in report filter area: Down, Then Over
- Report filter fields per column: 0

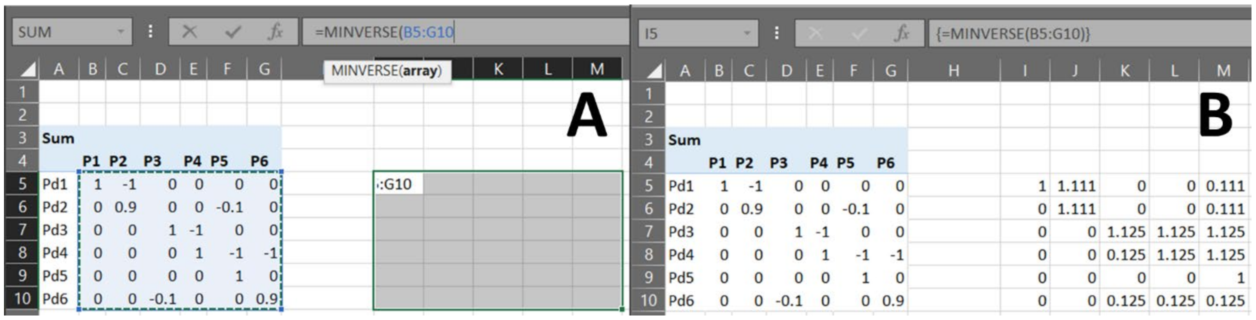
Format
- [ ] For error values show:
- [✓] For empty cells show: 0
- [✓] Autofit column widths on update
- [✓] Preserve cell formatting on update

Sum	P1	P2	P3	P4	P5	P6
Pd1	1	-1	0	0	0	0
Pd2	0	0.9	0	0	-0.1	0
Pd3	0	0	1	-1	0	0
Pd4	0	0	0	1	-1	-1
Pd5	0	0	0	0	1	0
Pd6	0	0	-0.1	0	0	0.9

Figure 4. Pivot Table configuration, settings, and output

2.2. Taking the Inverse

The next step within Excel is to generate the inverse matrix of $\mathbf{A}$, which is $\mathbf{A}^{-1}$ in Equation 1.6. Excel contains matrix functions, though they can be difficult to use. To use the matrix inversion function, MINVERSE, I must first select an array of precisely the correct size – the location of the output. Then, enter the function and select the original matrix (shown in Panel A of Figure 5). Finally, use the keys **control + shift + enter** to apply the equation to the selected range. Note that these keys turn the selected range into an array formula. The result is shown in Panel B of Figure 5.



Panel A formula: `=MINVERSE(B5:G10`, selected range `:G10`

Panel B formula (cell I5): `{=MINVERSE(B5:G10)}`

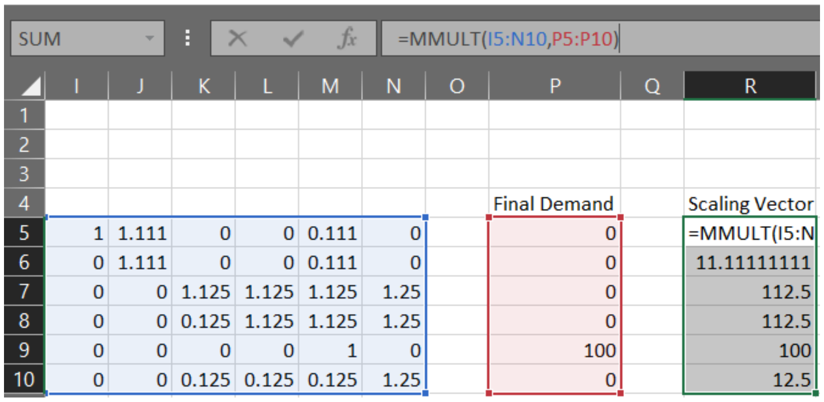
Sum	P1	P2	P3	P4	P5	P6
Pd1	1	-1	0	0	0	0
Pd2	0	0.9	0	0	-0.1	0
Pd3	0	0	1	-1	0	0
Pd4	0	0	0	1	-1	-1
Pd5	0	0	0	0	1	0
Pd6	0	0	-0.1	0	0	0.9

I	J	K	L	M
1	1.111	0	0	0.111
0	1.111	0	0	0.111
0	0	1.125	1.125	1.125
0	0	0.125	1.125	1.125
0	0	0	0	1
0	0	0.125	0.125	0.125

Figure 5. Creating an inverse matrix in Excel using the keys **control + shift + enter**

2.3. The Scaling Vector

Next, I will create a final demand vector and specify the final desired output. In this example, the final demand is 100 units of Pd_5. I will then highlight cells to be used for the scaling vector and will use the MMULT command to multiply $\mathbf{A}^{-1}F$, as shown in Figure 6. Again use **control + shift + enter** to generate the scaling vector.

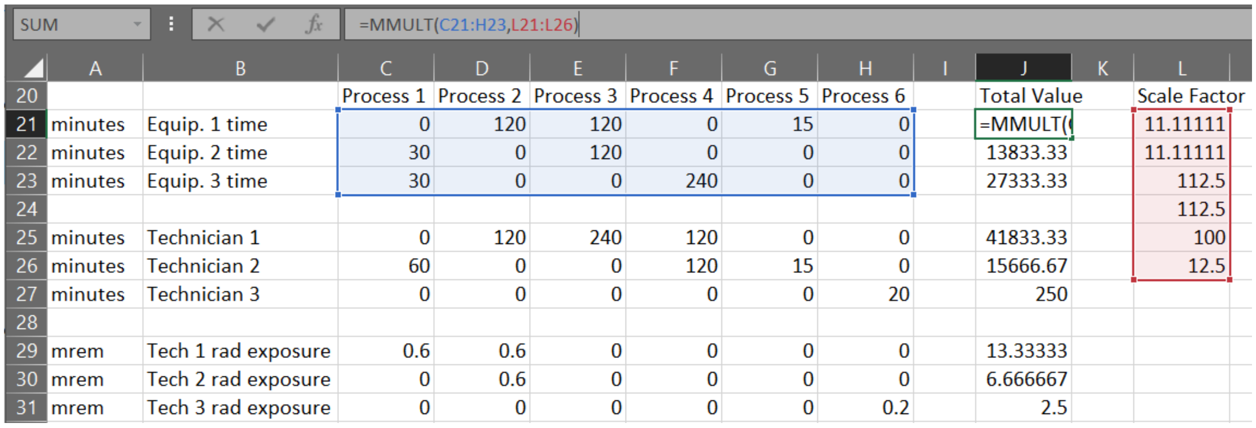

SUM | =MMULT(I5:N10,P5:P10)

	I	J	K	L	M	N	O	P	Q	R
1										
2										
3										
4								Final Demand		Scaling Vector
5	1	1.111	0	0	0.111	0		0		=MMULT(I5:N
6	0	1.111	0	0	0.111	0		0		11.11111111
7	0	0	1.125	1.125	1.125	1.25		0		112.5
8	0	0	0.125	1.125	1.125	1.25		0		112.5
9	0	0	0	0	1	0		100		100
10	0	0	0.125	0.125	0.125	1.25		0		12.5

Figure 6. Generating the scaling vector from the inverse process matrix and a final demand

2.4. The Auxiliary Matrix

Finally, I will set up relevant auxiliary matrices. There are three different auxiliary matrices in Figure 7: one to track equipment runtime, one to track technician worktime, and one to track radiation exposure. These could be combined into a single matrix – keeping them apart is mere personal preference.


SUM | =MMULT(C21:H23,L21:L26)

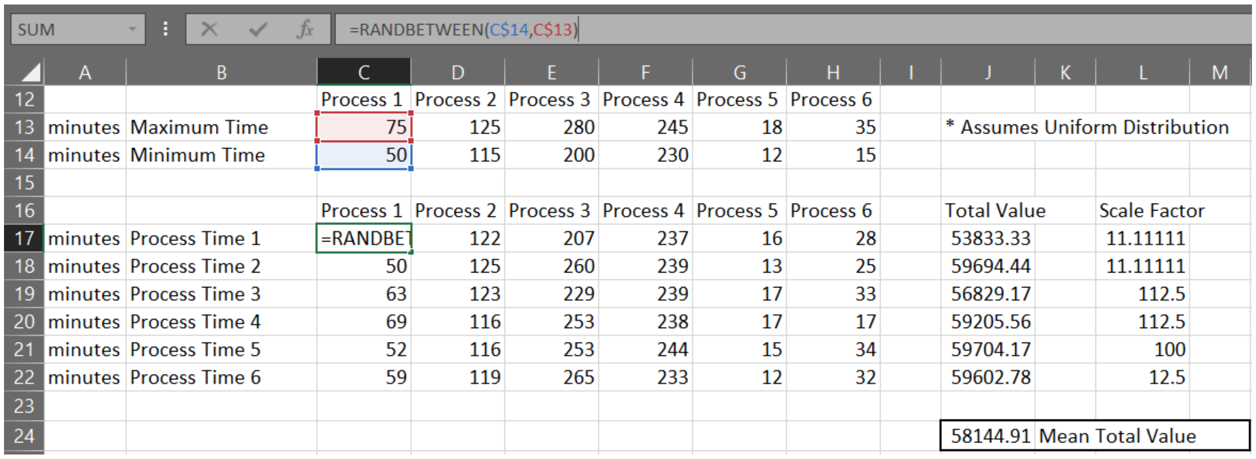
	A	B	C	D	E	F	G	H	I	J	K	L
20			Process 1	Process 2	Process 3	Process 4	Process 5	Process 6		Total Value		Scale Factor
21	minutes	Equip. 1 time	0	120	120	0	15	0		=MMULT((		11.11111
22	minutes	Equip. 2 time	30	0	120	0	0	0		13833.33		11.11111
23	minutes	Equip. 3 time	30	0	0	240	0	0		27333.33		112.5
24												112.5
25	minutes	Technician 1	0	120	240	120	0	0		41833.33		100
26	minutes	Technician 2	60	0	0	120	15	0		15666.67		12.5
27	minutes	Technician 3	0	0	0	0	0	20		250		
28												
29	mrem	Tech 1 rad exposure	0.6	0.6	0	0	0	0		13.33333		
30	mrem	Tech 2 rad exposure	0	0.6	0	0	0	0		6.666667		
31	mrem	Tech 3 rad exposure	0	0	0	0	0	0.2		2.5		

Figure 7. Using auxiliary matrices with the scaling vector to determine total values of various metrics

In order to calculate the total values for the various metrics, I will use the MMULT function, select the $\mathbf{B}$ matrix and the scaling factor, and use **control + shift + enter** as before. The outputs, shown in *Column J* of Figure 7, represent the total value for each row's metric after producing the final demand specified.

2.5. Implementing Basic Stochasticity

The first "level" of stochasticity discussed in Section 1.6 can be easily implemented in Excel. As a reminder, this "level" is uncertainty within the auxiliary metrics, but not the process itself. Figure 8 shows a basic implementation of this uncertainty. For each process, a uniform distribution with upper and lower bounds are defined in *Rows 13* and *14* of Figure 8. The RANDBETWEEN function is used across *Rows 17-22* to generate values. Calculation of totals proceeds as any other $\mathbf{B}$ matrix. The final step is to take the mean of *Column J* to determine the average process time, shown at the bottom right of Figure 8.



SUM

=RANDBETWEEN(C$14,C$13)

	A	B	C	D	E	F	G	H	I	J	K	L	M
12			Process 1	Process 2	Process 3	Process 4	Process 5	Process 6					
13	minutes	Maximum Time	75	125	280	245	18	35		* Assumes Uniform Distribution			
14	minutes	Minimum Time	50	115	200	230	12	15					
15													
16			Process 1	Process 2	Process 3	Process 4	Process 5	Process 6		Total Value		Scale Factor	
17	minutes	Process Time 1	=RANDBET	122	207	237	16	28		53833.33		11.11111	
18	minutes	Process Time 2	50	125	260	239	13	25		59694.44		11.11111	
19	minutes	Process Time 3	63	123	229	239	17	33		56829.17		112.5	
20	minutes	Process Time 4	69	116	253	238	17	17		59205.56		112.5	
21	minutes	Process Time 5	52	116	253	244	15	34		59704.17		100	
22	minutes	Process Time 6	59	119	265	233	12	32		59602.78		12.5	
23													
24										58144.91	Mean Total Value		

Figure 8. Basic stochasticity in Excel. Rows 13 and 14 define a uniform distribution while 17-22 implement the model

In this example, only six “runs” or instances are used for the Monte Carlo model. However, a modeler should likely use between 100 and 10,000 runs depending on the amount of uncertainty and ranges of distributions [3].

Conclusion

As demonstrated above, a matrix-based process model creates a concise representation of a process and enables calculation of many different metrics of interest. Furthermore, this model can be easily implemented in Excel. Excel allows for a level of transparency, edit-ability, and share-ability not often found in other modeling programs. The Excel model could be expanded upon, such as by creating an interactive dashboard sheet, to better control and utilize the methodology. This technique is best-suited for smaller processes, or instances where the customer requires a “quick” answer and can accept the assumptions and limitations described in Section 1.7. If the limitations are acceptable, this technique provides a good option for process modeling.

Works Cited

[1] P. D. Blair and R. E. Miller, Input-Output Analysis: Foundations and Extensions, United Kingdom: Cambridge University Press, 2009.

[2] M. Newman, Networks, United Kingdom: OUP Oxford, 2018.

[3] R. Heijungs and S. Suh, The Computational Structure of Life Cycle Assessment, Netherlands: Springer Netherlands, 2013.

[4] G. G. Brown, "Modeling," in *INFORMS Analytics Body of Knowledge*, United Kingdom, Wiley, 2018, pp. 155-198.

[5] S. R. Upreti, Process Modeling and Simulation for Chemical Engineers: Theory and Practice, Germany: Wiley, 2017.

[6] J. Sherwood, R. Clabeaux and M. Carbajales-Dale, "An extended environmental input–output lifecycle assessment model to study the urban food–energy–water nexus," *Environmental Research Letters,* vol. 12, no. 105003, 2017.

For more information, please contact:
John Sherwood, Ph.D.
Process Modeling and Analysis Group (E-2)
Los Alamos National Laboratory
Los Alamos, NM 87545
(505) 664-0295
jsherwood@lanl.gov